МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа № 5**

*по дисциплине «Алгоритмы и структуры данных»*

Выполнил студент

группы ПИбд-12

Нгуен Т. М.

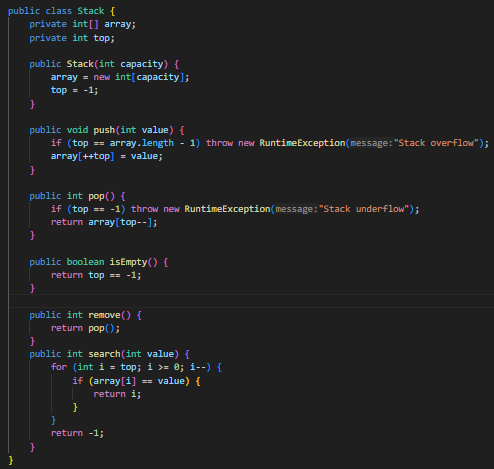
Проверил доцент кафедры

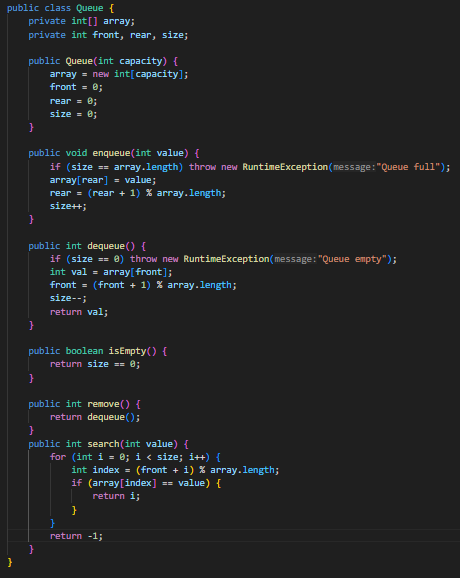
«Информационные системы»

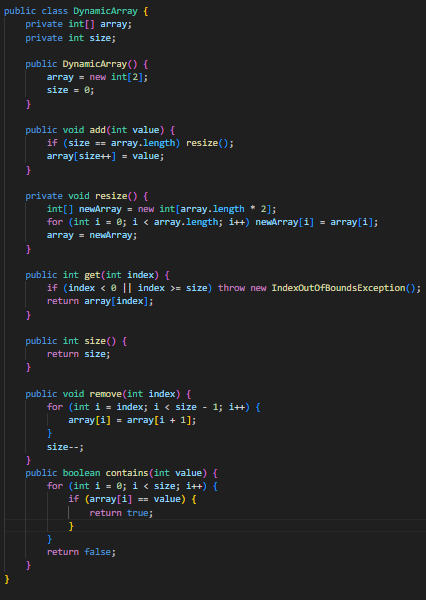
Кулешов А.В.

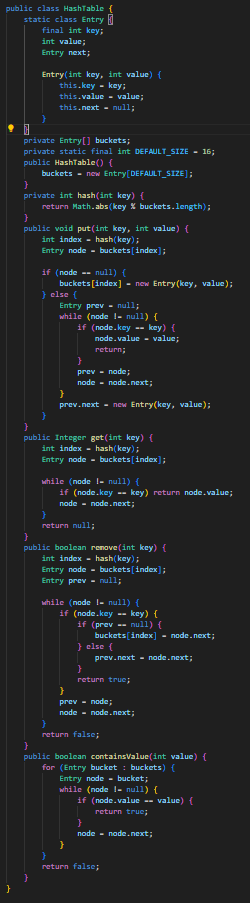
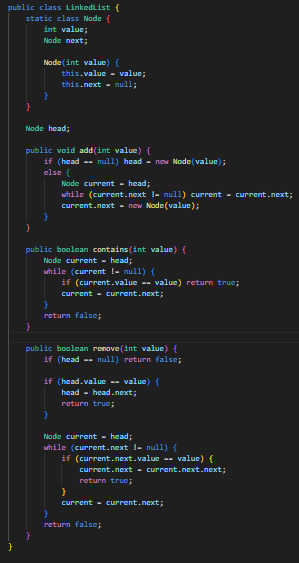
Ульяновск, 2025

  
Для этого я взял АТД – Стек и очередь  
Для Структуры данных – Динамический массив, связанный список и хеш-таблица







  
  
  
  
Цель – Сравнить структуры данных и АТД, из вставки, поиски и удаления и понять, когда лучше использовать каждый из них

**1. Связанный список (Linked List)**

**Когда использовать:**

* Когда часто вставляются и удаляются элементы **в середине или начале** списка.
* Когда размер коллекции **неизвестен заранее**.
* Когда **доступ по индексу не важен**.

**Когда не использовать:**

* Когда нужен **быстрый доступ по индексу**.
* Когда операции поиска выполняются часто (линейная сложность).

**Операции (для односвязного списка):**

| **Операция** | **Сложность** | **Комментарий** |
| --- | --- | --- |
| Вставка | O(1) в начало, O(n) в середину/конец | Требуется пройти список |
| Поиск | O(n) | Линейный перебор |
| Удаление | O(1) при известном узле, иначе O(n) |  |

**2. Хеш-таблица (Hash Table)**

**Когда использовать:**

* Когда нужен **быстрый доступ к данным по ключу**.
* Когда количество данных **велико и поиск критичен по времени**.

**Когда не использовать:**

* Когда нужен **упорядоченный** доступ.
* Когда **мало данных** или ключи плохо распределены (возможны коллизии).

**Операции (в среднем случае):**

| **Операция** | **Сложность** | **Комментарий** |
| --- | --- | --- |
| Вставка | O(1) | Может быть O(n) при коллизиях/resize |
| Поиск | O(1) | Быстрый доступ по хешу ключа |
| Удаление | O(1) | Аналогично поиску по хешу |

**3. Динамический массив (ArrayList в Java)**

**Когда использовать:**

* Когда нужен **быстрый доступ по индексу**.
* Когда элементы **чаще читаются, чем изменяются**.
* Когда **размер меняется**, но не слишком часто.

**Когда не использовать:**

* Когда часто требуются **вставки/удаления в середину или начало**.
* Когда нужен **частый resize**, т.к. это дорого.

**Операции:**

| **Операция** | **Сложность** | **Комментарий** |
| --- | --- | --- |
| Вставка | O(1) в конец (амортизированная), O(n) в начало/середину | Требуется сдвиг элементов |
| Поиск | O(1) | По индексу |
| Удаление | O(n) | Сдвиг элементов после удаления |

**4. Стек (Stack)**

**Когда использовать:**

* Когда требуется **LIFO** поведение (последний пришёл — первый вышел).
* Для задач типа **обратной польской нотации**, **обратного хода**, **анализатора выражений**, **рекурсии**.

**Когда не использовать:**

* Когда нужен **доступ к произвольному элементу**.
* Когда важен **порядок обхода всех элементов**.

**Операции:**

| **Операция** | **Сложность** | **Комментарий** |
| --- | --- | --- |
| Вставка (push) | O(1) | В конец |
| Поиск | O(n) | Линейный поиск |
| Удаление (pop) | O(1) | Удаляется верхний элемент |

**5. Очередь (Queue)**

**Когда использовать:**

* Когда требуется **FIFO** поведение (первый пришёл — первый вышел).
* Для **планировщиков задач**, **буферов**, **обработки событий**.

**Когда не использовать:**

* Когда требуется **доступ к произвольному элементу**.
* Когда нужно **удаление не с начала**.

**Операции:**

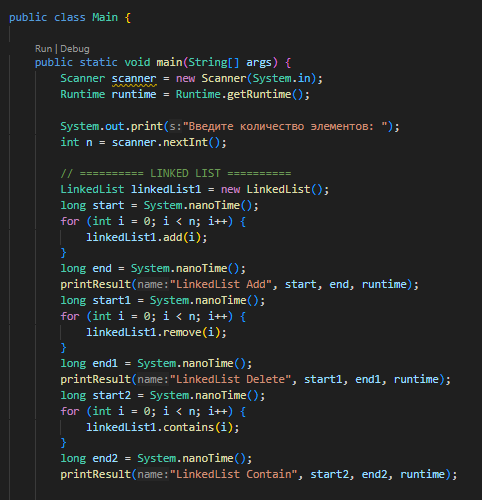
| **Операция** | **Сложность** | **Комментарий** |
| --- | --- | --- |
| Вставка (enqueue) | O(1) | В конец очереди |
| Поиск | O(n) | Линейный |
| Удаление (dequeue) | O(1) | С начала очереди |

**Таблица-сравнение по операциям**

| **Структура** | **Вставка** | **Поиск** | **Удаление** | **Особенности** |
| --- | --- | --- | --- | --- |
| Связанный список | O(1)/O(n) | O(n) | O(1)/O(n) | Хорош для вставки, плох для поиска |
| Хеш-таблица | O(1) | O(1) | O(1) | Быстрая, но без порядка |
| Динамический массив | O(1)/O(n) | O(1) | O(n) | Быстрый доступ, плохая вставка |
| Стек | O(1) | O(n) | O(1) | LIFO |
| Очередь | O(1) | O(n) | O(1) | FIFO |

В теории таблица выглядит так, но на практике та же хеш таблица на самом деле занимает добавление минимум O(n), если размерность данных большая, потому что хеш таблица имеет ограниченное количество мест и нужно делать пробивания и увеличивать размер

Для динамического после удаления, нужно делать сдвиг всего массива, что затрудняет время работы

Для точного сравнения скорости действий у каждого их них я реализовал такой участок кода  
  
Тут для связанного списка, у остальных аналогичная логика работы  
Вот результат в виде графиков  
  
Выглядит некрасиво, связанный список запорол все, по-этому отдельно сделаю таблицу, без связанного списка, чтобы удобнее сравнить